L'analyse de la complexité en temps de cet algorithme se fait principalement sur la base de trois fonctions : **read_data()**, **bag_algorithm()** et **analyse()**.

La fonction **read_data()**: Elle lit un fichier CSV et effectue quelques opérations de manipulation de données. Ces opérations comprennent l'ajout d'une nouvelle colonne, la filtration de lignes et le tri.

Le temps d'exécution de cette fonction dépendra principalement de la taille du fichier CSV (supposons **n** lignes). La complexité en temps de cette fonction peut être considérée comme

O(n log n)

à cause de l'opération de tri, qui est généralement l'opération la plus coûteuse en temps ici.

La fonction **bag_algorithm()**: Cette fonction parcourt chaque ligne du DataFrame filtré et effectue quelques opérations conditionnelles et arithmétiques. Supposons que le DataFrame filtré ait **m** lignes. La complexité en temps de cette fonction est donc **O(m)**.

La fonction **analyse()**: Elle crée un nouveau DataFrame à partir d'un dictionnaire, puis calcule la somme de certaines colonnes. La complexité en temps de cette fonction est également **linéaire**, soit **O(m)**, car elle doit parcourir chaque élément du dictionnaire/du DataFrame.

La fonction principale **main()** exécute ces trois fonctions pour chaque fichier dans my_data_files. Supposons qu'il y ait **p** fichiers. Par conséquent, la complexité en temps totale serait

O(p*(n log n + 2m)).

Cependant, sans plus d'information sur la relation entre **n** et **m** (c'est-à-dire le nombre de lignes avant et après le filtrage), il est difficile de simplifier davantage cette expression.

Pour la complexité en espace :

La fonction **read_data()**: Crée un DataFrame basé sur le fichier CSV, donc la complexité en espace est **O(n)**, où **n** est le nombre de lignes dans le fichier CSV.

La fonction **bag_algorithm()**: Remplit un dictionnaire avec certaines actions. Dans le pire des cas, cela pourrait être toutes les actions, donc la complexité en espace est **O(m)**, où **m** est le nombre de lignes dans le DataFrame filtré.

La fonction **analyse()**: Crée un nouveau DataFrame à partir du dictionnaire, donc la complexité en espace est également **O(m)**.

La complexité en espace totale de l'algorithme est donc **O(n + m)**, où **n** est le nombre total de lignes dans tous les fichiers CSV, et **m** est le nombre total de lignes dans tous les DataFrames filtrés.

Notez que ces estimations de complexité supposent que chaque opération effectuée par pandas (par exemple, l'ajout d'une colonne, le filtrage des lignes, le tri) a une **complexité linéaire** ou **n log n.**

Cependant, les détails réels peuvent varier en fonction de l'implémentation de ces opérations dans pandas. En outre, ces estimations ne tiennent pas compte du temps nécessaire pour lire les fichiers CSV du disque, ce qui pourrait être significatif pour de grands fichiers.

Pour optimiser ce code, vous pourriez envisager de réduire le nombre de fois où vous parcourez le DataFrame ou le dictionnaire. Par exemple, vous pourriez combiner certaines des opérations dans la fonction bag_algorithm() et analyse() pour ne parcourir les données qu'une seule fois. De plus, si vous pouvez éviter de trier les données dans read_data(), cela pourrait également améliorer le temps d'exécution. Cependant, toutes ces suggestions dépendent des exigences spécifiques de votre programme.